Directive blocks

Directives are brought in by modules—if you activate a new module, a specific set of directives becomes available. Modules may also enable *directive blocks*, which allow for a logical construction of the configuration:

```
events {
    worker_connections 1024;
}
```

The `events` block that you can find in the default configuration file is brought in by the *Events module*. The directives that the module enables can only be used within that block—in the preceding example, `worker_connections` will only make sense in the context of the `events` block. On the other hand, some directives must be placed at the root of the configuration file, because they have a global effect on the server. The root of the configuration file is also known as the `main` block.

For the most part, blocks can be nested into each other, following a specific logic. The following sequence demonstrates the structure of a simple website setup, making use of nested blocks:

```
http {
    server {
        listen 80;
        server_name example.com;
        access_log /var/log/nginx/example.com.log;
        location ^~ /admin/ {
            index index.php;
        }
    }
}
```

The topmost directive block is the `http` block, in which you may declare a variety of configuration directives as well as one or more `server` blocks. A `server` block allows you to configure a virtual host, in other words, a website that is to be hosted on your machine. The `server` block, in this example, contains some configuration that applies to all HTTP requests with a `Host` header exactly matching `example.com`.

Within this `server` block, you may insert one or more `location` blocks. These allow you to enable settings only when the requested URI matches the specified path. More information is provided in the *The Location block* section of *Chapter 3, HTTP Configuration*.

Last but not least, configuration is inherited within children blocks. The `access_log` directive (defined at the `server` block level in this example) specifies that all HTTP requests for this server should be logged into a text file. This is still true within the `location` child block, although you have the option of disabling it by reusing the `access_log` directive:

```
[…]
    location ^~ /admin/ {
        index index.php;
        access_log off;
    }
[…]
```

In this case, logging will be enabled everywhere on the website except for the `/admin/` location path. The value set for the `access_log` directive at the `server` block level is overridden by the one at the `location` block level.

Advanced language rules

There are a number of important observations regarding the Nginx configuration file syntax. These will help you understand certain language rules that may seem confusing if you have never worked with Nginx before.

Directives accept specific syntaxes

You may indeed stumble upon complex syntaxes that can be confusing at first sight, like the following one:

```
rewrite ^/(.*)\.(png|jpg|gif)$ /image.php? file=$1&format=$2 last;
```

Syntaxes are directive-specific. While the `root` directive only accepts a simple character string defining the folder containing the files that should be served for a website, the `location` block and the `rewrite` directive support complex expressions in order to match particular patterns. Some other directives such as `listen` accept up to 17 different parameters. Syntaxes will be explained along with directives in their respective chapters.

Later on, we will detail a module (the *Rewrite* module) which allows for a much more advanced logical structure through the `if`, `set`, `break`, and `return` blocks and directives, and the use of variables. With all these new elements, configuration files will begin to look like programming scripts. You will find that, the more modules we discover, the richer the syntax becomes.